A Multi-objective Genetic Algorithm to Rank State-Based Test Cases

Lionel Briand¹, Yvan Labiche², and Kathy Chen²

¹SnT Centre, University of Luxembourg, Luxembourg
lionel.briand@uni.lu
²Systems and Computer Engineering, Carleton University, Ottawa, Canada
labiche@sce.carleton.ca, kathy_h_chen@hotmail.com

Abstract. We propose a multi-objective genetic algorithm method to prioritize state-based test cases to achieve several competing objectives such as budget and coverage of data flow information, while hopefully detecting faults as early as possible when executing prioritized test cases. The experimental results indicate that our approach is useful and effective: prioritizations quickly achieve maximum data flow coverage and this results in early fault detection; prioritizations perform much better than random orders with much smaller variance.

Keywords: State-based testing, Prioritization, Multi-objective optimization, Genetic algorithm.

1 Introduction

The earlier defects are detected the better. During testing this can be achieved by prioritizing test case executions: test cases with higher (estimated) defect detection capabilities are executed earlier than others. Such a prioritization (or ranking) also allows one to stop test case execution when the budget dedicated to this activity is exhausted.

In this paper, we address the problem of ordering the execution of black-box test cases. Specifically, we rank state-based test cases derived according to the well-known transition tree method [1] as it has been shown to be a good compromise among available selection criteria [2] (e.g., all-transition pairs, all-transitions [3]). Prioritization is performed according to the data-flow the test cases cover in the test model since this relates to fault detection [4], and tester-defined constraints: goal to reach in terms of data-flow coverage and/or maximum budget (cost) of the execution of prioritized test cases. As a result, finding a ranking is a multi-objective optimization problem, which we solve with a multi-objective genetic algorithm. The solution is an optimal test sequence in the sense that it aims to find as many defects as possible as early as possible when executing prioritized test cases. We evaluate how optimal is approach on a data structure class that exhibits a state-based behavior.

Section 2 discusses related work. Section 3 describes our multi-objective genetic algorithm. Section 4 reports on a case study. Conclusions are drawn in Section 5.
2 Related Work

Our work relates to state-based testing, data-flow identification from operation contracts, and test case prioritization.

**State-based testing** consists in devising transition sequences to exercise different elements of the state model, as stated by a selection criterion [3]. Several alternative criteria exist [1, 3], including all transitions, all transition pairs, full predicates (an adaptation of MC/DC to guards), and round-trip path (transition) tree. Binder’s round-trip path tree [1] adapts Chow’s W method [5] to UML state machines, and has been shown empirically to be a good compromise in terms of cost and effectiveness (at detecting faults) between all transitions and all transition pairs (or full predicate) [2], even though effectiveness depends on the way the tree has been built [6]. The reader interested in more details on this criterion is referred to the above literature.

To ascertain the state reached by an implementation at the end of a test case execution and use that information in a test oracle, one can rely on the state invariant (when the system state is observable, as in our case) or other characteristics of the state machine as in protocol conformance testing [5, 7].

One of the main challenges of state-based testing is the path feasibility problem, that is, determining whether a transition sequence derived according to a selection criterion is feasible (i.e., one can find test inputs for the sequence such that it executes). This problem, akin to the path sensitization problem in white-box testing [8], is known to be un-decidable [9]. We assume test cases (i.e., transition sequences) are feasible and we focus on prioritizing them.

**Operation contracts** include pre- and post-conditions. In UML, they can be expressed using the OCL [10]. Briand et al. [4] provided rules for identifying definitions and uses of model elements from OCL operation contracts and transition guard conditions, and applied those rules to UML state machine diagrams. Then, they used well-known criteria such as all-definitions and all-DU-paths to define test cases. One important observation they made on three different case studies is that the set of DU-paths equals the set of DU-pairs in the test model obtained when applying their strategy (i.e., only one definition-clear path from the definition to the use for each DU-pair). It is therefore sufficient to determine DU-pairs to cover DU-paths. This may not be the case in general though, and only further studies will confirm (or not) this result. They also noticed that the data flow analysis they propose can be used as an indicator of the defect revealing power of a test suite. A general rule is that in a set of alternative transition trees derived from a state machine diagram, the transition tree covering the most data flow information (in the model) has a better capability to detect defects. Since the other data-flow analysis methods for UML state machines were less complete (e.g., support the UML notation to a lesser extent) [4], we adopted the rules defined by Briand et al.

**Test case prioritization** has received a lot of attention in two areas of software testing [11]: ranking system level test cases, and prioritization during regression testing. In both cases the objective is to rank the test cases such that the most beneficial ones (according to some criterion) are executed first. Criteria for prioritization are varied, e.g., coverage (e.g., code, model element, requirements), priority, criticality.